Divide and Conquer

การแบ่งแยกและเอาชนะ (Divide and Conquer) เป็นวิธีการหาแก้ปัญหาโดยการแบ่งปัญหาที่ต้องการแก้เป็นปัญหาย่อย (subproblem) หลายๆ ปัญหา โดยที่ปัญหาย่อยเหล่านั้นต้องมีความเหมือนกับปัญหาแรกที่ต้องการแก้ เพียงแต่มีขนาดเล็กกว่า หลังจากนั้นจึงหาคำตอบของแต่ละปัญหาย่อย แล้วนำแต่ละคำตอบมารวมกันเพื่อเป็นคำตอบของปัญหาใหญ่ อัลกอริทึมการแบ่งแยกและเอาชนะ เป็นดังนี้

```
1  solveDC(P) // P: problem
2  {
3     if (P is trivial)
4        return Solve(P)
5     else
6        divide P into P1, P2, …, Pk
7        for i = 1 to k
8           Si = SolveDC(Pi) // S: Solution
9        S = combine(S1, S2, …, Sk)
10       return S
11 }
```

วิธีการแบ่งแยกและเอาชนะจะใช้ประโยชน์ของโครงสร้างหรือความสัมพันธ์แบบวนซ้ำ (Recursive Structure or Relation) มาแก้ปัญหาย่อยเหล่านั้น จากอัลกอริทึมจะประกอบไปด้วย 3 ขั้นตอนหลัก คือ

1. Divide เป็นการแบ่งปัญหาออกเป็นปัญหาย่อย หรือที่เรียกว่า subprogram ที่มีลักษณะเหมือน ๆ กัน
2. Conquer เป็นการหาคำตอบของแต่ละปัญหาย่อยแบบเวียนบังเกิด
3. Combine นำคำตอบย่อยที่ได้มารวมกัน

อัลกอริทึมมาตรฐานสำหรับ divide and conquer ได้แก่

- Quick sort
- Merge sort
- Karatsuba algorithm
- Strassen algorithm
- อัลกอริทึมจากปัญหาแนว computational geometry ได้แก่
  - Convex hull
  - Closet pair of points

ในส่วนของ time complexity

```
void solve(int n){
    if(n==0)
        return;
    solve(n/2);
    solve(n/2);
    for(int i=0; i<n; i++){
        //some constant time operations
    }
}
```

Time complexity ของการใช้อัลกอริทึม divide and conquer เป็น time ในส่วนของ recurrence relation คือ $T(n) = 2T\left(\frac{n}{2}\right) + n$

Asymptotic time complexity = $O(nlogn)$

***

Decrease and conquer

- บางครั้งไม่จำเป็นต้องทำปัญหาให้เป็น subproblem หลาย ๆ subproblem แต่ทำปัญหาให้มีขนาดเล็กลงเป็นเพียง one smaller subproblem
- มักจะเรียกว่า decrease and conquer เช่น Binary search

Binary Search

Problem ต้องการค้นหาข้อมูล โดยชุดของข้อมูลทั้งหมดต้องเรียงลำดับ

จากขั้นตอนการแก้ปัญหาตามวิธีการแบ่งแยกและเอาชนะสามารถทำได้ดังนี้

1. Divide: อินพุตของปัญหา คือ ลำดับของข้อมูลที่เรียงจากน้อยไปมาก (สมมติให้ลำดับมีขนาด n หน่วย) และข้อมูลที่เราต้องการค้นหาเรียกว่า key ในอัลกอริทึมนี้จะแบ่งปัญหาให้เป็นปัญหาย่อยที่มีขนาดเป็นครึ่งหนึ่งของปัญหาเดิม คือ $\frac{n}{2}$, $\frac{n}{4}$, $\frac{n}{8}$, ..., 1 หน่วย ตามลำดับ
2. Conquer: ปัญหาขนาด 1 หน่วยคือปัญหาที่มีขนาดเล็กสุด ซึ่งเราสามารถแก้ปัญหานี้ได้โดยตรง โดยการเปรียบเทียบ key กับข้อมูลที่อยู่ในลำดับที่มีเพียง 1 ตัว ถ้าข้อมูลทั้งสองมีค่าเดียวกัน ก็สามารถคืนตำแหน่งของข้อมูลที่พบได้ ถ้าข้อมูลทั้งสองมีค่าต่างกัน ก็คืนค่า 0 เพื่อแสดงว่า key ไม่อยู่ในลำดับที่โจทย์ให้
3. Combine: ในขั้นนี้เราจะนำผลลัพธ์จากปัญหาย่อยขนาดเล็กที่สุดมาเป็นคำตอบของปัญหาแรก

key = 19 กำหนดให้ low = 1 และ high = 8

รอบที่ 1 mid = 4

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
5	6	7	8	12	15	19	25

low = 5

รอบที่ 2

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
5	6	7	8	12	15	19	25

low = 7

รอบที่ 3

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
5	6	7	8	12	15	19	25

key = list[7]

อัลกอริทึมการค้นหาข้อมูลแบบไบนารีเป็นดังนี้

แบบที่ 1 อัลกอริทึมแบบวนซ้ำ

```
1  binary_search(list, key)
2     low = 0, high = size(list)-1
3     while low <= high
4        mid = (low + high)/2
5        if list[mid] == key
6           return mid
7        else if key<list[mid]
8           high = mid-1
9        else
10          low = mid+1
       return -1 // target was not found
```

Time complexity = $O(\log n)$

Space complexity = $O(1)$

แบบที่ 2 อัลกอริทึมแบบ recursive

Algorithm

1. **Base case:** the array is empty, return false
2. Compare the key to the element in the middle of the array
3. If it's equal, then we found the key and we return the position of the key in the array
4. If it's less, then the key must be in the left half of the array
   4.1 Binary search the element (recursively) in the left half
5. If it's greater, then the key must be in the right half of the array
   5.1 Binary search the element (recursively) in the right half

```
1  BinarySearch(list[low..high], key)
2  {
3     if (low > high)
4        return -1
5     else
6        mid = (low + high)/2
7        if (key == list[mid])
8           return mid     // bass case
9        else
10          if (key < list[mid])
11             return BinarySearch(list[low..mid-1], key)
12          else
13             return BinarySearch(list[mid+1..high], key)
14 }
```

Time complexity $= T(n) = T\left(\frac{n}{2}\right) + 1 = O(\log n)$

Space complexity = $O(\log n)$

Binary Search

จงเขียนโปรแกรม Binary Search (1) แบบวนซ้ำ (2) แบบเรียกตัวเอง (recursion function)

ข้อมูลนำเข้า บรรทัดที่ 1 จำนวนเต็ม K แทนข้อมูลที่ต้องการค้นหา

บรรทัดที่ 2 จำนวนเต็ม N แทนจำนวนข้อมูลทั้งหมด

บรรทัดที่ 3 ถึง บรรทัดที่ N+2 ประะกอบด้วยจำนวนเต็ม N จำนวน คั่นด้วยช่องว่าง

ข้อมูลนำออก บรรทัดที่ 1 กรณีพบข้อมูล ให้แสดงตำแหน่งของข้อมูล

กรณีไม่พบข้อมูลแสดงข้อความ "NOT FOUND"

บรรทัดที่ 2 แสดงข้อมูลที่ได้เรียงลำดับจากน้อยไปมากโดยคั่นด้วยช่องว่าง

ตัวอย่าง

Input	**Output**
19 8 8 12 7 15 6 19 5 25	7 5 6 7 8 12 15 19 25
30 8 8 12 7 15 6 19 5 25	NOT FOUND 5 6 7 8 12 15 19 25

Maximum SubArray Sum โดยใช้ Divide and Conquer

กำหนด 1D-array ที่อาจจะประกอบด้วยจำนวนเต็มบวกและจำนวนเต็มลบ ให้หาผลรวมของสมาชิกใน subarray ที่มีค่ามากที่สุด (the largest sum)

ตัวอย่าง

กำหนด {-2,-5,**6,-2,-3,1,5**,-6}

maximum subarray sum คือ 7

คำอธิบาย maximum subarray sum เป็นผลบวกของสมาชิกในลำดับที่ต่อเนื่องกันคือ **6,-2,-3,1,5**

แบบ naïve method (หรือ Brute force algorithm)

ให้ Time complexity เท่ากับ $O(n^2)$

ใช้ลูปซ้อน 2 ลูป โดยลูปชั้นนอกเริ่มจากตัวแรก และลูปชั้นใน ดำเนินการหาค่า maximum sum ที่เป็นไปได้จาก element ตัวแรก แล้วเปรียบเทียบค่า sum ทุก ๆ ค่าสำหรับแต่ละ element หลัง element แรกก็จะได้ค่ามากสุดสำหรับ ลูปนอกตัวแรก วนจนครบทุก element (ทำหน้าที่เป็นตำแหน่งแรกใน subarray) เมื่อทำจนครบก็จะได้ overall maximum sum ออกมา

```
Algorithm maximumSum(A: array of integers ,n: size of A array):
Input : An n-element array A of numbers
Output: The maximum sum of subarray A of numbers.
max_sum ← INT_MIN
for i←0 to n-2 do
      sum←0 //for each sum of sub-array A[i..j]
      for j←i to n-1 do
            sum ← sum + A[j]
            if (sum>max_sum)then
                  max_sum ← sum
return max_sum
```

แบบ Divide and conquer algorithm ให้ Time complexity เท่ากับ $O(nlogn)$
มีขั้นตอนการทำงานดังนี้

1. แบ่งครึ่งอาร์เรย์ออกเป็นสองส่วน
2. Return the maximum of following three
    - Maximum subarray sum in left half (Make a recursive call)
    - Maximum subarray sum in right half (Make a recursive call)
    - Maximum subarray sum such that the subarray crosses the midpoint

การหา maximum subarray sum ฝั่งซ้ายกับขวาสามารถใช้ recursive calls แต่การหา maximum subarray sum ที่ข้าม midpoint (mid) ให้หา maximum sum ที่เริ่มต้นจาก midpoint และสิ้นสุดที่จุดทางด้านซ้ายของกึ่งกลางนั้น (low) แล้วหาค่า maximum sum เริ่มต้นจากจุด mid+1 และสิ้นสุดที่จุดทางด้านขวา(high) สุดท้ายทำการรวมและส่งค่า maximum subarray sum ออกมา

```
Algorithm maximumSum(A: array of integers ,low: the position on left
of midpoint, high: the position on right of the midpoint):
Input : An n-element array A of numbers
Output: The maximum sum of subarray A of numbers.

//if the array contains only one element
if high == low then
      return A[low] // base case

//find the middle array element
mid ← (low+high)/2

//find the maximum sum subarray for the left subarray
//including the middle element
left_max ← INT_MIN
sum ← 0
for i ← mid to low do
      sum ← sum+A[i]
      if sum>left_max then
            left_max ← sum

//find the maximum sum subarray for the right subarray
//excluding the middle element
right_max ← INT_MIN
sum ← 0
for i ← mid+1 to high do
      sum ← sum + A[i]
      if (sum>right_max)then
            right_max ← sum
//recursively find the maximum subarray sum for the left and right
subarray, and take maximum
max_left_right ← max(maximumSum(A,low,mid),maximumSum(A,mid+1,high))

return max(max_left_right, left_max+right_max)
```

การใช้ **Kadane's algorithm** สำหรับหาค่า Maximum sum subarray ใน 1D-array ให้ Time complexity ดีกว่าใช้ divide and conquer เพราะให้ **time complexity เท่ากับ** $\boldsymbol{O(n)}$

DC01: Maximum Subarray Sum

ให้นักเรียนเขียนโปรแกรมเพื่อแก้ปัญหา `maximum subarray sum` โดยใช้ `Divide and Conquer`

ข้อมูลนำเข้า บรรทัดที่ 1 จำนวนเต็ม `n` แทนขนาดของอาร์เรย์
บรรทัดที่ 2 ถึงบรรทัดที่ `n+1` ประกอบด้วยจำนวนเต็ม `n` จำนวน

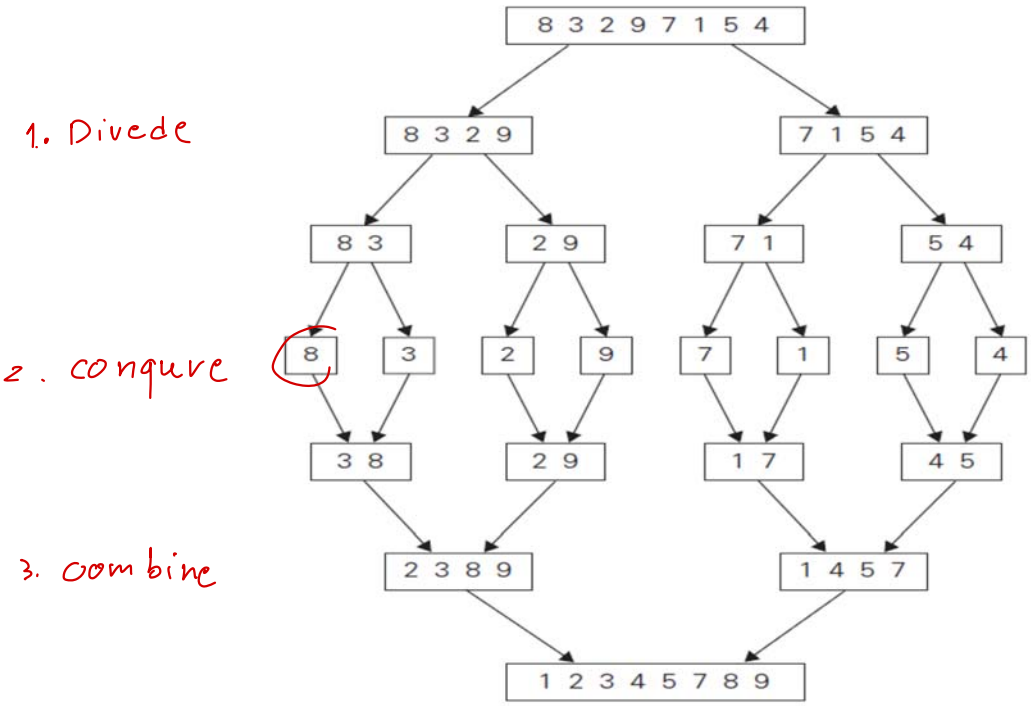
ข้อมูลนำออก ผลรวมของสมาชิกใน `subarray` ที่มีค่ามากที่สุด

ตัวอย่าง

Input	Output
8 -2 -5 6 -2 -3 1 5 -6	7
7 2 -4 1 9 -6 7 -3	11

Merge Sort

Problem ต้องการเรียงลำดับข้อมูล



การเรียงลำดับข้อมูลแบบ Merge sort สามารถแบ่งขั้นตอนการแก้ปัญหาตามวิธีการแบ่งแยกและเอาชนะได้ดังนี้

1. Divide: อินพุตเป็น ลำดับของข้อมูล มีขนาด n หน่วย (ข้อมูลในลำดับซ้ำกันได้) อัลกอริทึมจะแบ่งปัญหาให้เป็นสองปัญหาย่อยที่มีขนาดเป็นครึ่งหนึ่งของปัญหาเดิมจนมีขนาดเป็น 1 หน่วย
2. Conquer: ในการแก้ปัญหา หากปัญหาย่อยมีขนาดเกิน 1 ปัญหาจะถูกแบ่งครึ่งเป็นปัญหาย่อยต่อไปอีก แต่ถ้าปัญหาย่อยมีขนาด 1 หน่วยแล้ว ปัญหาจะได้รับการแก้ได้โดยตรง เพราะการเรียงลำดับของข้อมูล 1 ตัวคือการคืนค่าข้อมูลนั้น
3. Combine: เป็นขั้นตอนการนำผลลัพธ์ของสองปัญหาย่อยมารวมกัน (Merge) ผลลัพธ์ของแต่ละปัญหาย่อยก็คือลำดับที่ถูกเรียงลำดับมาแล้ว นำมารวมกัน

อัลกอริทึมของ Merge Sort เป็นดังนี้

```
1   MergeSort(list[low..high])
2   {
3      if (low >= high)
4         return list    // base case
5      else
6         mid = (low + high)/2
7         MergeSort(list[low..mid])
8         MergeSort(list[mid+1..high])
9         Merge(list[low..high], mid)
10  }
11
12  Merge(list[low..high], mid)
13  {
14     temp = new array[low..high]
15     i = low, j = mid+1, k = 0
16     while(i <= mid and j <= high)
17     {
18        if(list[i] < list[j])
19           temp[k++] = list[i++]
20        else
21           temp[k++] = list[j++]
22     }
23     while(i <= mid) //copy remaining low list
24        temp[k++] = list[i++]
25     while(j <= high)//copy remaining high list
26        temp[k++] = list[j++]
27
28     return list[low..high] = t[low..high]
    }
```

recusive

DC02: Merge Sort

ให้นักเรียนโปรแกรม Merge Sort

ข้อมูลนำเข้า บรรทัดที่ 1 จำนวนเต็ม N แทนจำนวนข้อมูลทั้งหมด
บรรทัดที่ 2 ถึง บรรทัดที่ N+2 ประะกอบด้วยจำนวนเต็ม N จำนวน
คั่นด้วยช่องว่าง

ข้อมูลนำออก บรรทัดที่ 1 แสดงข้อมูลก่อนการเรียง
บรรทัดที่ 2 แสดงข้อมูลที่ได้เรียงลำดับจากน้อยไปมากโดยคั่นด้วยช่องว่าง

ตัวอย่าง

Input	Output
8 8 3 2 9 7 1 5 4	1 2 3 4 5 7 8 9

Longest Common Prefix

กำหนดเซตของสตริงให้ จงหา the longest common prefix

ตัวอย่าง

```
Input  : {“mission”, “missed”, “misspell”, “misshaped”}
Output : "miss"

Input  : {"apple", "ape", "april"}
Output : "ap"
```

แบบ Brute Force algorithm

```
Algorithm commonPrefix(A: array of strings ,n: size of A array):
Input : An n-element array A of strings
Output: The longest common prefix
Function LCP(word1,word2)
//compare word1 and word2 for each duplicate character
return  prefix string in both word1 and word2

prefix ← A[0]
for i←1 to n-1 do
     prefix ← LCP(prefix,A[i]);
return prefix
```

Time complexity = $O(NM)$ เมื่อ N คือจำนวนคำทั้งหมด และ M คือความยาวของคำที่ยาวที่สุด

Longest Common Prefix using Divide and Conquer

แนวคิด

แบ่งสตริงเป็นสองกลุ่ม (two halve) แล้วทำ recursive ทั้งสองกลุ่ม คล้ายกับ merge sort ยกเว้นว่าเราหา **LCP ของ two halves** แทนที่จะ merge เข้าด้วยกัน

```
Algorithm findLCP(A: array of strings ,low, high):
// base case: if `low` is more than `high`, return an empty string
if (low > high) then
      return string("")

// base case: if `low` is equal to `high`, return the current string
if (low == high) then
      return words[low]  // base case

// find the mid-index
mid = (low + high) / 2

// partition the problem into subproblems and recur for each subproblem
X = findLCP(words, low, mid)   // ซ้าย
Y = findLCP(words, mid + 1, high) // ขวา

// return the longest common prefix of strings `X` and `Y`
return LCP(X, Y)
```

Time complexity = $O(NM)$

Space complexity = $O(M \log N)$ สำหรับการเรียกใช้สแตกเพื่อทำ recursive

DC03: Longest Common Prefix

จงเขียนโปรแกรมหา `longest common prefix`

ข้อมูลนำเข้า บรรทัดที่ 1 จำนวนคำ N โดยที่ $2 \leq N \leq 1000$

บรรทัดที่ 2 ถึง `N + 1` เป็นคำที่มีความยาว M โดยที่ $2 \leq M \leq 80$

ข้อมูลนำออก `longest common prefix`

กรณีไม่มี `common prefix` ให้พิมพ์คำว่า "`none`"

ตัวอย่าง

Input	**Output**
4 mission missed misspell misshaped	miss
5 monitor monitory monster monstrous monument	mon
3 monday tuesday friday	none

Closest pair of points

Finding closest pair of points

กำหนดเซตของจุด n จุด คือ $(x_1, y_1),\ (x_2, y_2), \dots, (x_n, y_n)$ บนระนาบ
ให้หาจุดสองจุดใด ๆ $(x_i, y_i), (x_j, y_j)$ ที่ให้ระยะห่างระหว่างจุด 2 จุดดังกล่าวใกล้ที่สุด มักใช้การหาระยะห่างแบบ Euclidean distance เพื่อคำนวณระยะห่าง โดยสูตรคำนวณ คือ $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$
*****all points have distinct x-coordination and y-coordination**

ปัญหานี้เป็นปัญหาที่นำไปประยุกต์กับการควบคุมการจราจรทางอากาศ เช่น ใช้สำหรับการติดตาม (monitor) เครื่องบินว่าเข้าใกล้กันและมีโอกาสเกิดชนกันขึ้นได้

แบบ Brute force โดยตรวจสอบระยะห่างของจุดทุก ๆ คู่ ที่กำหนดให้ โดยมีจำนวนการคำนวณระยะห่างระหว่างจุดและคู่จุดที่ต้องเปรียบเทียบคือ $C(n, 2) = n(n-1)/2$
ดังนั้น Time complexity = $O(n^2)$

Algorithm: Brute-Force Algorithm for closet pair of Points

function closest-pair$((x_1, y_1),\ (x_2, y_2), \dots, (x_n, y_n))$:pairs of real numbers)
$min\ =\ \infty$
$for\ i = 2\ to\ n$
    $for\ j = 1\ to\ i - 1$
        $if\ (x_j - x_i)^2 + (y_j - y_i)^2 < min\ then$
            $min = (x_j - x_i)^2 + (y_j - y_i)^2$
            $closest\ pair = ((x_i, y_i), (x_j, y_j))$
$return\ closest\ pair$

แบบ Divide and Conquer
time complexity = $O(nlogn)$

วิธีคิด

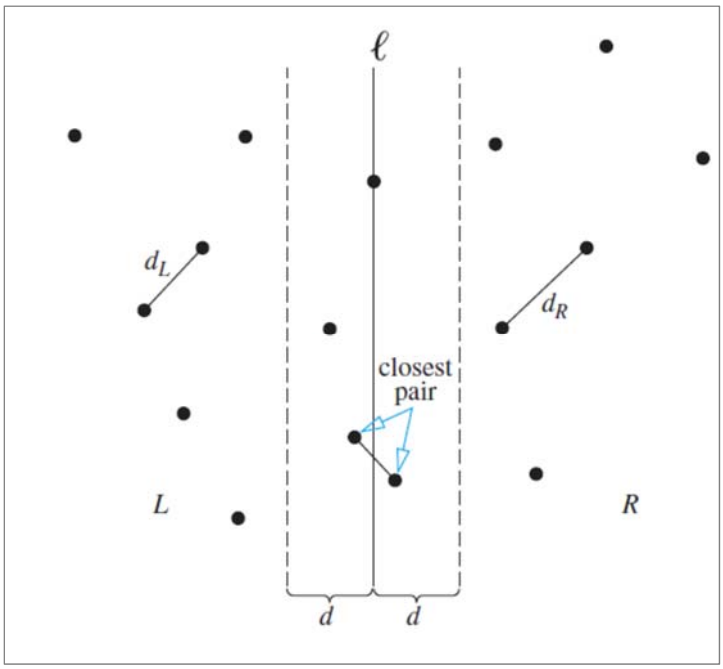
1. ถ้า n=2 แสดงว่ามีจุด 1 คู่ ดังนั้นระยะห่างระหว่างจุดคู่นี้เป็น ระยะห่างใกล้สุด (minimum distance)
2. เรียงจุดที่กำหนดให้ตาม x-coordinates จากน้อยไปมาก เก็บไว้ในลิสต์ แทนด้วย Px และเรียงจุดที่กำหนดให้ตาม y-coordinates จากน้อยไปมาก แทนด้วย Py (ในขั้นตอนการเรียงอาจใช้ merge sort หรือ quick sort ดังนั้น time complexity = $O(nlogn)$
3. ใช้ลิสต์ในขั้น (2) สำหรับขั้นตอน recursive
4. ในส่วนของ recursive algorithm แบ่งปัญหาออกเป็น 2 ส่วน ซึ่งแต่ละส่วนประกอบด้วยจุดครึ่งหนึ่ง โดยใช้ลิสต์ของจุดที่เรียงตาม x-coordinates แล้วสร้างเส้นในแนวตั้ง l เพื่อแบ่งจุด n

จุด เป็น 2 ส่วนเป็นด้านซ้ายและด้านขวาเท่า ๆ กัน ดังรูปที่ 1 ดังนั้นแต่ละด้านจะมีจุด n/2 จุด เป็นผลให้ขั้นตอนของ recursive ไม่ต้องเรียงจุดตาม x-coordinates อีกครั้ง เพราะเราสามารถเลือก sorted subset of all the points ขั้นตอนนี้ใช้ time complexity = $O(n)$

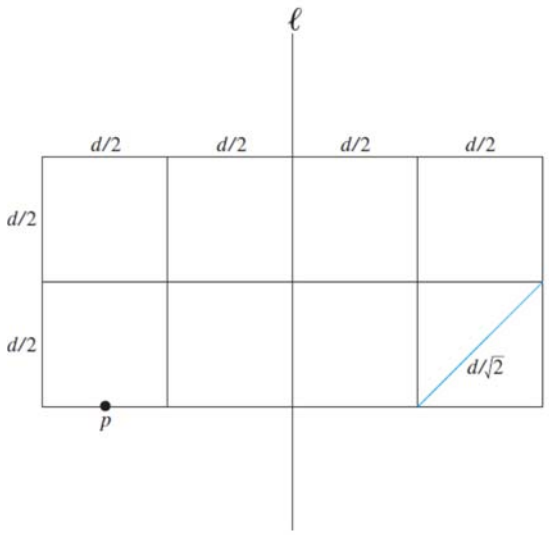
5. มี 3 ประเด็นที่ต้องพิจารณา คือ
   1) ทั้งสองจุดอยู่ในพื้นที่ L
   2) ทั้งสองจุดอยู่ในพื้นที่ R
   3) มีจุดหนึ่งอยู่ด้าน L และอีกจุดอยู่ด้าน R

   ดังนั้นใน recursive อัลกอริทึมคำนวณ d_L และ d_R โดยที่ d_L แทนระยะห่างน้อยที่สุดระหว่างจุดในพื้นที่ L และ d_R แทนระยะห่างน้อยที่สุดระหว่างจุดในพื้นที่ R แล้วคำนวณ $d = \min(d_L, d_R)$
6. เพื่อให้ขั้นตอนการแบ่งปัญหาสำเร็จ (divide) ในส่วนของ conquer อัลกอริทึมทำงานโดยพิจารณาประเด็นที่สองจุด (closest points) อยู่คนละพื้นที่ จุดหนึ่งอยู่ L อีกจุดอยู่ R ดังนั้นถ้าสองจุดที่อยู่คนละพื้นที่เป็น closest points ระยะห่างระหว่างจุดต้องน้อยกว่า d
7. จากรูปสร้างลิสต์ strip เพื่อเก็บจุดที่อยู่ในพื้นที่ที่ห่างจาก l เป็นระยะ d ทั้งซ้ายและขวา โดยเลือกมาจากลิสต์ในข้อ 2 ที่เรียงลำดับตาม y-coordinates

   Time complexity ในขั้นตอนนี้เป็น $O(n)$ เกิดจากการเปรียบเทียบ
8. พิจารณาแต่ละจุดที่อยู่ในลิสต์ strip คำนวณระยะห่างระหว่างจุดไปยังทุกจุดใน strip เป็นจุดที่ให้ค่าระยะห่างน้อยกว่า d
9. <u>สังเกต</u> สำหรับแต่ละจุด p ดังรูปที่ 2 เนื่องจากระยะห่างที่น้อยสุดเป็น d ดังนั้นเมื่อพิจารณาจุด p ใน strip จะไม่มีจุด 2 จุดในพื้นที่ 1 ช่องตาราง โดยสามารถมีจุดได้ 1 จุดเท่านั้นเพราะ $\frac{d}{\sqrt{2}} < d$
10. หาระยะห่าง d' ใน strip แล้ว $closest\ pair\ points = \min(d', d)$



รูปที่ 1 การทำงานแบบ recursive ของอัลกอริทึมสำหรับ closest pairs problem ขั้นตอน Divide



รูปที่ 2 มีจุดได้มากที่สุด 7 จุด สำหรับแต่ละจุด p ที่ต้องพิจารณาในลิสต์ strip

Closest pair of points algorithm

1. sort all points according to x coordinates.
2. Divide all points in two halves.
3. Recursively find the smallest distances in both subarrays.
4. Take the minimum of two smallest distances. Let the minimum be d.

5. Create an array strip[] that stores all points which are at most d distance away from the middle line dividing the two sets.
6. Find the smallest distance in strip[].
7. Return **the minimum of d** and **the smallest distance** calculated in above step 6.

Total running time:

- divide set of points in half each time: $O(\log n)$ depth recursive
- Merge takes $O(n)$ times
- Recurrence: $T(n) \leq 2T\left(\frac{n}{2}\right) + cn$
- Same as Merge Sort: $O(nlogn)$ time

DC04: Closest pair of points

จงเขียนโปรแกรมเพื่อแก้ปัญหา Closest pair of points

ข้อมูลนำเข้า บรรทัดที่ 1 จำนวนคำ N โดยที่ $2 \leq N \leq 1000$

บรรทัดที่ 2 ถึง N + 1 เป็นจำนวนเต็มสองจำนวน x แทน x-coordinate และ y แทน y-coordinate ของจุด

ข้อมูลนำออก closest pair of points distance โดยระบุเป็นทศนิยม 4 ตำแหน่ง

กรณีไม่มี common prefix ให้พิมพ์คำว่า "none"

ตัวอย่าง

Input	Output
6 2 3 12 30 40 50 5 1 12 10 3 4	1.4142
5 0 0 0 1 100 45 2 3 9 9	1.0000
5 0 0 -4 1 -7 -2 4 5 1 1	1.4142

สามารถฝึกเพิ่มเติม

- UVA 10245 "The Closest Pair Problem" [difficulty: low]
- SPOJ #8725 CLOPPAIR "Closest Point Pair" [difficulty: low]
- CODEFORCES Team Olympiad Saratov - 2011 "Minimum amount" [difficulty: medium]
- Google CodeJam 2009 Final " Min Perimeter "[difficulty: medium]
- SPOJ #7029 CLOSEST "Closest Triple" [difficulty: medium]
- TIMUS 1514 National Park [difficulty: medium]

การ Implement จาก geeksforgreeks

1. เรียงลำดับจุดทั้งหมดตาม x-coordinates แทนด้วย Px และเรียงลำดับจุดทั้งหมดตาม y-coordinates แทนด้วย Py
2. แบ่งจุดเป็น 2 ส่วน (ครึ่ง-ครึ่ง)
3. หาระยะทางสั้นสุดแบบ recursive ทั้งสอง subarray
4. หาค่า minimum of the smallest distances กำหนดให้เป็น d
5. สร้างอาร์เรย์ชื่อ strip[] ที่เก็บ all points ซึ่ง at most d distance away from the middle line dividing by two sets
6. หา the smallest distance in strip[]
7. return the minimum of d และ the smallest distance ที่คำนวณได้ในข้อ 6

```
#include<iostream>
#include<float.h>
#include<stdlib.h>
#include<math.h>
using namespace std;

struct Point{
    int x,y;
};

//need for sorting
int compareX(const void* a, const void* b){
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

int compareY(const void* a, const void* b){
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

float dist(Point p1, Point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

//if there are only 2 points
float bruteForce(Point P[],int n){
    float min = FLT_MAX;
    for (int i=0;i<n;i++)
        for(int j=i+1;j<n;++j)
            if(dist(P[i],P[j])<min)
                min = dist(P[i],P[j]);
    return min;

}

float min(float x,float y){
```

```
    return (x<y)? x:y;
}

float stripClosest(Point strip[], int size,float d){
    float min = d;

    for(int i=0; i<size; ++i)
        for(int j = i+1;j<size && (strip[j].y-strip[i].y)<min;++j)
            if(dist(strip[i],strip[j])<min)
                min = dist(strip[i],strip[j]);

    return min;

}

float closestUtil(Point Px[],Point Py[], int n){
    if (n<3) return (bruteForce(Px,n));

    //find the middle point
    int mid = n/2;
    Point midPoint =Px[mid];

    Point Pyl[mid];
    Point Pyr[n-mid];

    int li = 0, ri = 0;
    for(int i=0;i<n;i++)
    {
        if(Py[i].x <= midPoint.x && li<mid)
            Pyl[li++]=Py[i];
        else
            Pyr[ri++]=Py[i];
    }

    float dl = closestUtil(Px,Pyl,mid);
    float dr = closestUtil(Px+mid,Pyr,n-mid);

    float d = min(dl,dr);

    Point strip[n];
    int j = 0;
    for(int i=0;i<n;i++)
        if(abs(Py[i].x-midPoint.x)<d)
            strip[j] = Py[i], j++;

    return stripClosest(strip,j,d);
}

float closest(Point P[],int n){
    Point Px[n];
    Point Py[n];

    for(int i=0;i<n;i++){
        Px[i] = P[i];
        Py[i] = P[i];
```

```
    }

    qsort(Px,n,sizeof(Point),compareX);
    qsort(Py,n,sizeof(Point),compareY);

    return closestUtil(Px,Py,n);

}

int main(){
    Point P[]={{2,3},{12,30},{40,50},{5,1},{12,10},{3,4}};
    int n =sizeof(P)/sizeof(P[0]);

    cout<<"The smallest distance is "<<closest(P,n);

    return 0;
}
```

DC05: Big Mod

กำหนดให้

$$R := B^P mod\ M$$

ข้อมูลนำเข้า บรรทัดที่ 1 จำนวนเต็ม N แทนจำนวนชุดทดสอบ
สามบรรทัดถัดไป ถึงบรรทัดที่ $3 \times (N + 1)$ แสดงแต่ละชุดทดสอบประกอบด้วย จำนวนเต็ม B, P และ M ตามลำดับ โดยที่ $0 \leq B, P \leq 2147483647$ และ $1 \leq M \leq 46340$

ข้อมูลนำออก แต่ละบรรทัดแสดงผลลัพธ์ของแต่ละชุดทดสอบ

ตัวอย่าง

Input	Output
3 3 18132 17 17 1765 3 2374859 3029382 36123	13 2 13195

คำแนะนำ
ใช้ความสัมพันธ์

$$(x \times y)\ mod\ m = \big((x\ mod\ m) \times (y\ mod\ m)\big)\ mod\ m$$

DC06: Fibonacci Words

ลำดับ Fibonacci นิยามได้ดังนี้

$$fibo_1 = 1$$

$$fibo_2 = 1$$

$$fibo_n = fibo_{n-2} + fibo_{n-1}$$

ดังนั้นเราจะได้ลำดับดังนี้ $1, 1, 2, 3, 5, 8, 13, 21, \ldots$

หากเรานิยามลำดับ Fibonacci โดยเริ่มต้นจากจำนวนอื่น เช่น

$$f_1 = 5$$

$$f_2 = 4$$

$$f_n = f_{n-2} + f_{n-1}$$

ลำดับที่ได้คือ $5, 4, 9, 13, 22, 35, 57, \ldots$

ถ้านิยามลำดับ Fibonacci ที่ไม่ใช่จำนวน ดังนี้

$$s_1 = N$$

$$s_2 = A$$

$$s_n = s_{n-2} + s_{n-1}$$

โดยเครื่องหมาย $+$ ในที่นี้แทนการนำค่าสตริงมาต่อกัน ลำดับที่ได้เป็นดังนี้
$N, A, NA, ANA, NAANA, \ldots$ เรียกลำดับนี้ว่า ***Batmanacci***

คำสั่ง ให้นักเรียนเขียนโปรแกรมเพื่อหาอักขระตัวที่ K^{th} ที่อยู่ในลำดับที่ N^{th} ของลำดับ ***Batmanacci*** โดยให้ค่า N และ K

<u>ข้อมูลนำเข้า</u>

บรรทัดที่ 1 จำนวนเต็ม T แทนจำนวนชุดทดสอบ โดยที่ $1 \leq T \leq 10$

สำหรับแต่ละชุดทดสอบ

บรรทัดที่ 2 ถึงบรรทัดที่ $N+1$ จำนวนเต็ม N และ K คั่นด้วยช่องว่างหนึ่งช่อง โดยที่ N แทนลำดับสตริงตัวที่ N^{th} ของลำดับ *Batmanacci* โดยที่ $1 \leq N \leq 10^5$ และ K แทนอักขระตัวที่ K^{th} ของลำดับสตริงตัวที่ N^{th} ของลำดับ *Batmanacci* โดยที่ $1 \leq K \leq 10^{18}$

ข้อมูลนำออก สำหรับแต่ละชุดทดสอบ ให้พิมพ์อักขระตัวที่ K^{th} ของลำดับสตริงตัวที่ N^{th} ของลำดับ *Batmanacci*

ตัวอย่าง

Input	Output
2 7 7 777 777	N A

คำแนะนำ

เพื่อให้รองรับ $1 \leq K \leq 10^{18}$ ให้ใช้ข้อมูลชนิด

unsigned **long long** int 0 to 18,446,744,073,709,551,615